How did we get here?

- In the beginning, security was by physical isolation (1950-1963)
  - Easy: You bring your data, control the machine, take everything away
  - Still do this today with VMs and crypto (+ enclaves if VM host is untrusted)

- Timesharing brought the basic dilemma of security: (1963-1982)
  - Isolation vs. sharing
    - Hard: Each user wants a private machine, isolated from others
    - but users want to share data, programs and resources

- Since then, things have steadily gotten worse (1982-2015)
  - Less isolation, more sharing, no central management
  - More valuable stuff in the computers
  - Continued misguided search for perfection (following the NSA’s lead)
Wisdom

■ If you want security, you must be prepared for inconvenience.
  —General B.W. Chidlaw, 12 December 1954

■ When it comes to security, a change is unlikely to be an improvement.
  —Doug McIlroy, ~1988

■ The price of reliability is the pursuit of the utmost simplicity.
  It is a price which the very rich find most hard to pay.
  —Tony Hoare, 1980 (cf. Matthew 19:24)

■ But who will watch the watchers? She’ll begin with them and buy their silence.
  —Juvenal, sixth satire, ~100
What we know how to do

- Secure something simple very well
- Protect complexity by isolation and sanitization
- Stage security theatre

What we don’t know how to do

- Make something complex secure
- Make something big secure if it’s not isolated
- Keep something secure when it changes
- Get users to make judgments about security
- Understand privacy—fortunately not an SOSP topic
## Themes

- **Goals**: Secrecy (confidentiality), integrity, availability  (CIA: Ware 1970)
- **Gold standard**: Authentication, authorization, auditing  (S&S 1975)
- **Principals**: People, machines, programs, …  (Dennis 1966, DEC 1991)
- **Groups/roles**: make policy manageable  (Multics 1968, NIST 1992)

### Oppositions

<table>
<thead>
<tr>
<th>Winner</th>
<th>Loser</th>
<th>(in deployment, not good vs. bad)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convenience</td>
<td>vs. Security</td>
<td></td>
</tr>
<tr>
<td>Sharing</td>
<td>vs. Isolation</td>
<td></td>
</tr>
<tr>
<td>Bug fixes</td>
<td>vs. Correctness</td>
<td></td>
</tr>
<tr>
<td>Policy/mechanisms</td>
<td>vs. Assurance</td>
<td></td>
</tr>
<tr>
<td>Access control</td>
<td>vs. Information flow</td>
<td></td>
</tr>
<tr>
<td>Timeline</td>
<td>Themes</td>
<td>Systems</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1960s</td>
<td><strong>Timesharing</strong>: ACLs; access control matrix; VMs; passwords; capabilities; domains; gates</td>
<td><strong>CTSS</strong>; <strong>Multics</strong>; <strong>CP/CMS</strong>; <strong>Cal TSS</strong>; <strong>Adept-50</strong>; <strong>Plessey 250</strong></td>
</tr>
<tr>
<td>1970s</td>
<td><strong>TS</strong>: LANs/Internet (e/e security); public key; multi-level sec.; ADTs/objects; least privilege; Trojans; isolation by crypto; amplification; undecidability</td>
<td><strong>Unix</strong>; <strong>VMS</strong>; <strong>VM/370</strong>; <strong>IBM RACF</strong>; <strong>Clu</strong>; <strong>Hydra</strong>; <strong>Cambridge CAP</strong></td>
</tr>
<tr>
<td>1980s</td>
<td><strong>Workstations</strong>; <strong>client/server</strong>: Orange Book; global authentication; Clark and Wilson</td>
<td><strong>A1 VMS</strong>; <strong>SecureID</strong>; <strong>Morris worm</strong>; <strong>IX</strong></td>
</tr>
<tr>
<td>1990s</td>
<td><strong>PCs</strong>; <strong>Web</strong>: sandboxes; Java security; crypto export; decentralized information flow; Common Criteria; biometrics; RBAC; BAN; SFI; SET</td>
<td><strong>Browsers</strong>; <strong>SSL</strong>; <strong>NT</strong>; <strong>Linux</strong>; <strong>PGP</strong>; <strong>Taos</strong></td>
</tr>
<tr>
<td>2000s</td>
<td><strong>Web</strong>; <strong>JavaScript</strong>: buffer overflows; DDoS</td>
<td><strong>TPM</strong>; <strong>LSM</strong>; <strong>SELinux</strong>; <strong>seL4</strong>; <strong>HiStar</strong></td>
</tr>
<tr>
<td>2010s</td>
<td><strong>Web</strong>; <strong>big data</strong>: enclaves; homomorphic crypto</td>
<td><strong>Singularity</strong>; <strong>CryptDB</strong>; <strong>Ironclad</strong> ...</td>
</tr>
</tbody>
</table>
Foundation: Isolation

- A host isolates an execution environment
  - The basis for any security. Must trust the host
- Many ways to do it (and many bugs):

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java/JavaScript sandboxing</td>
<td>JVM/JS engine</td>
</tr>
<tr>
<td>Modules/objects</td>
<td>language/runtime</td>
</tr>
<tr>
<td>Software fault isolation</td>
<td>process</td>
</tr>
<tr>
<td>Processes</td>
<td>OS</td>
</tr>
<tr>
<td>Virtual machines</td>
<td>hypervisor</td>
</tr>
<tr>
<td>Limited comm (wires or crypto)</td>
<td>network</td>
</tr>
<tr>
<td>Air gaps: physical separation</td>
<td>physics</td>
</tr>
</tbody>
</table>

Host (CLR, kernel, hardware, VMM, ...)

- Java 1995
- Clu 1974
- Wahbe et al 1993
- CTSS 1962
- CP/40 1966
- DESNC 1985
- 1950; Tempest ~1955
Safe Sharing: Access Control

1. **Isolation boundary** limits attacks to channels (no bugs)
2. **Access Control** for channel traffic
3. **Policy** sets the rules

![Diagram](image)

- **Authentication**
  - Subject / Principal
  - Request

- **Authorization**
  - Guard / Reference monitor
  - Object / Resource

1. Isolation boundary
2. Access control
3. Policy

Host (CLR, kernel, hardware, VMM, ...)

Anderson report 1972
Access Control: The Gold Standard

 Authenticate principals: Who made a request?

- People, but also channels, servers, programs
  (encryption implements channels, so the key is a principal)

 Authorize access: Who is trusted with a resource?

- Group principals or resources, to simplify management
  - Can define a group by a property, e.g. “type-safe” or “safe for scripting”

 Audit requests: Who did what when?
Policy: What sharing is allowed?

- The guard evaluates a function: permissions = policy(subject, object)
  - If functions are too mathematical, call it an access matrix (Lampson 1971)

<table>
<thead>
<tr>
<th>Subject/principal</th>
<th>Object/resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>File foo</td>
<td>Database payroll</td>
</tr>
<tr>
<td>Alice</td>
<td>read, write</td>
</tr>
<tr>
<td>Bob</td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>write paychecks</td>
</tr>
</tbody>
</table>

- Permissions kept at the object are ACLs; at the subject, capabilities
  - Capabilities work for short term policy
    - File descriptors/handles in OS; objects in languages (Unix/Windows; Java, C#)
  - ACLs work for long-term policy; tell you who can access the resource
    - Groups of subjects and objects keep this manageable (Multics 1968)
Keeping Secrets: Information Flow Control

0. **Labels** on information
1. **Isolation boundary** limits flows to channels
2. **Flow control** based on labels
3. **Policy** says what flows are allowed

---

**Diagram Description**

- **0. Labels**
  - Data + Label
  - Guard / Ref mon
  - Send
  - Subject / Principal

- **1. Isolation boundary**
- **2. Flow control**
- **3. Policy**

**Others**

- **Authorization**
- **Authentication**

**Notes**

- Adept-50 1969
- Orange Book 1985

**References**

- Lampson: Perspectives on security
Information Flow Control

- Invented to model military classification (Adept-50 1969)
  - Label every datum: top secret/nuclear $\geq$ top secret $\geq$ secret
    - Labels form a lattice, and propagate: If $d_1$ is input to $d_2$, then $d_2$’s label is $\geq d_1$’s
  - Enforce with access control: label subjects, containers (Bell/LaPadula 1973)
    - No read up, write down; can be dynamic or static (Adept-50; Denning 1976)

- Decentralized flow control (Myers and Liskov 1998)
  - Anyone can invent labels. If you own a label, you can declassify it
    - Can do this in a language or in an OS (Jflow 1999; HiStar 2006)

- So far, none of this has been practical

- And then there are covert (side) channels: timing, radiation, power ...
  - Abstractions don’t keep secrets (Tempest 1955, Lampson 1972)
Who controls policy? DAC, MAC, RBAC

How to decide:
- Is the user or the program **malicious**? Insiders, Trojan horses
- Is the user **competent**? Policy can be tricky
- Is the user **motivated**? Security gets in the way of work and play

- Discretionary access control (DAC) : the object’s owner (CTSS 1963)
- Mandatory access control (MAC) : an administrator (1969; 1985)
  - MAC ≠ flow control
- Role based access control (RBAC): the app designer (NIST 1992)
  - Administrator just populates the roles
Distributed Systems: Cryptography

- **Communicate**, so need secure channels
  - Host, secure wire, …, but usually cryptography: General, **end to end**

- Basic crypto functionality: mathematical magic that implements:
  - **Sign** with $K^{-1}$/ verify with $K$: integrity
  - **Seal** with $K$ / unseal with $K^{-1}$: secrecy

  You can only do it if you know the key

- This gives you an **end-to-end** secure channel

- Public key crypto: $K \neq K^{-1}$; I can sign, anyone can verify  
  (RSA 1977)

- **Homomorphic** crypto: compute on encrypted data  
  (Gentry 2009)

  - This is too slow, but you can *simulate* it  
  (CryptDB 2011)

- Zero knowledge and **verifiable** computation  
  (Pinocchio 2013)
Distributed Systems: Understanding Trust

- Decentralized, so have to reason about trust, justifying by proofs

- Principals: people, machines, programs, services, protocols, …

- Accountability: principal says statement
  - Alice says read from file Foo

- Trust: principal A speaks for principal B
  - Alice says Bob@microsoft speaks for Alice
  - Microsoft says Key63129 speaks for Bob@microsoft
  - Key63129 says read from file Foo

- file Foo says Alice speaks for file Foo ACL entry
  - So Foo says read from file Foo
    - End to end reasoning

DEC 1989, 1991

3 October 2015

Lamson: Perspectives on security
Does it actually work? Assurance (Correctness)

- Keep it simple—Trusted Computing Base (TCB) (Rushby 1981)
  - One way is a security kernel: apps are not in the TCB. Works for sharing hardware

- Ideally, you **verify**: prove that a system satisfies its security spec
  - This means that *every* behavior of the system is allowed by the spec
    - Not the same as proving that it does everything in the manual
  - Today in seL4, Ironclad, … First tried in Gypsy (late 1970s)
  - What if the spec is wrong? Keep it simple

- Usually verifying is too hard, so you **certify** instead
  - Through some “independent” agency. Alas, process trumps substance
    - First by DoD for Orange Book, later international Common Criteria (1985, 1999)

- Or you can verify **some** properties: isolation, memory/type safety

- Or you can apply bandaids

3 October 2015
Lampson: Perspectives on security
No guarantees, but at least the bad guy has to work harder

- **Firewalls** to keep intruders out, look for suspicious traffic  
  (DEC 1988)
- **Signature** hacks to detect malware  
  (~1990)
- **Memory safety** hacks to catch writes outside array bounds  
  (Phrack 1996)
- **Intrusion detection** hacks to look for anomalous behavior  
  (SRI 1986)
- **Control Flow Integrity** to block jumps not in the normal flow  
  (MSR 2005)
- **Taint tracking** to keep unsanitized input away from execution  
  (CMU 2005)
- **Process** to enforce use of the tools  
  (MS SDL 2004)

“I don’t have to outrun the bear; I just have to outrun you.”

- These are not bad things, but they are hacks
What about *my* system? Configuration (Policy)

- If the code is correct, the configuration may still be wrong
  - You write the code once, but every system has its own configuration
  - It’s boring, and it changes. So either it’s small, or it’s wrong.
    - But it’s not small; there’s always another feature, another plausible scenario
      - There are 12 levels of indirection in Windows printing, each with its own security

- And configuration is usually done by amateurs
  - With MAC and RBAC at least it’s done by pros

**Conflict:** want fine grained policy, but can only manage coarse grain
  - Solution (never adopted): Lower aspirations, budget for complexity
What has worked? What hasn’t?

Worked ~ gotten wide adoption

**Worked**
- VMs
- SSL
- Passwords
- Safe languages
- Firewalls
- Process—SDL

**Failed**
- “Secure systems”
- Capabilities (except short term)
- Metrics for security
- MLS/Orange book
- User education
- Intrusion detection
Why don’t we have “real” security?

A. People don’t buy it
- Danger is small, so it’s OK to buy features instead
- Security is expensive
  - Configuring security is a lot of work
  - Secure systems do less because they’re older
- Security is a pain
  - It stops you from doing things
  - Users have to authenticate themselves
- Goals are unrealistic, ignoring technical feasibility and user behavior

B. Systems are complicated, so they have bugs
- Especially the configuration
What next?

- Lower aspirations. In the real world, good security is a bank vault
  - Hardly any computer systems have anything like this
  - We only know how to make simple things secure

- Access control doesn’t work—40 years of experience says so
  - Basic problem: its job is to say “No”
    - This stops people from doing their work, and then they relax the access control
    - Usually too much, but no one notices until there’s a disaster

- Retroactive security: focus on things that actually happened
  - Rather than all the many things that might happen
  - Real world security is retroactive
    - Burglars are stopped by fear of jail, not by locks
    - The financial system’s security depends on undo, not on vaults